Reprint Courtesy of International Business Machines Corporation, © International Business Machines Corporation

- **IBM Copyright Permission #21953**

OpenDoc/SOM and OLE/COM

Berthold Reinwald

Purpose of the talk

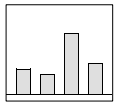
- Compound Documents: look & feel
- Component Software: what's behind the windows
- Technology Relationships: overview & preview
- OpenDoc/SOM and OLE/COM comparison
- Your first Component: "get down and dirty"
- Statement about the Applications

OpenDoc/OLE – Behind the Windows?

OpenDoc

This is a compound document, demon–strating the irregular shapes in OpenDoc.

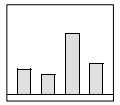
Judge yourself.

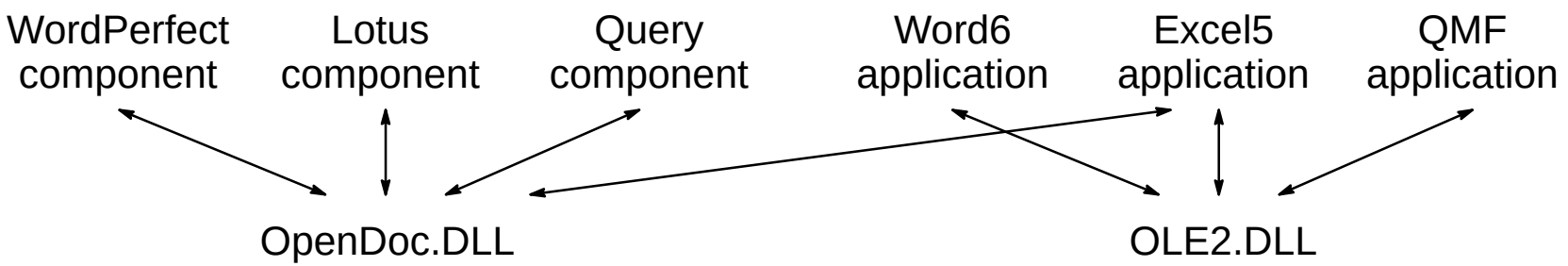


OLE

This is a compound document in OLE. Oh, regular shapes only.

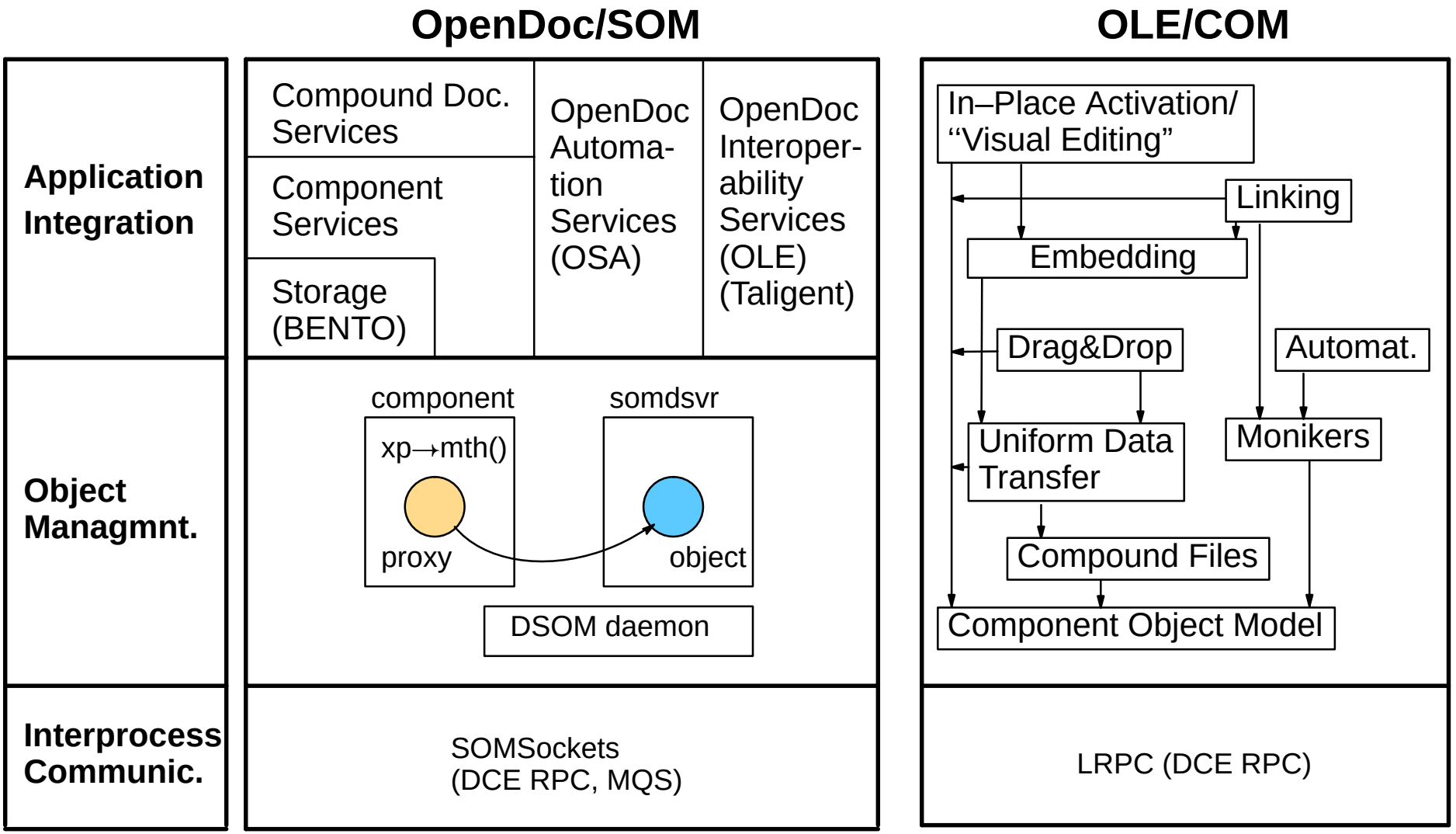
Judge yourself.





- a component/application is just a program executable or a DLL
- enabled software
- each component/application employs its own binary data (format)
- distributed objects
- remote method invocation

Technology Relationships



OpenDoc/SOM versus OLE/COM

	OpenDoc	OLE
Programming	SOM	COM
Development Effort	50 functions	126 functions
Scripting&Automat.	OSA, recordable macros	?
Activation Model	directly, across nested, multiple active objects	activate&edit, along nesting path, one at a time
Content Shape	irregularly	rectangular
Storage System	Bento	DocFiles (DOS FAT file)
Object Linking	persistent IDs	Moniker

	SOM	COM
Programming	object–oriented	object–based (no inherit.)
Languages	language–neutral (DTS)	language–neutral
Static Typing	statically typed	interfaces are static. typed
Distribution	DSOM	with Cairo
Method Resolution	offset, name, dispatch	interface vector table
Standards	OMG compliant (IDL, DTS)	MS proprietary
Platforms	OS/2, AIX, Windows	Windows, NT

Hello Component: HELLO.IDL

```
...
interface Hello : ContainerPart
{
   ...
   //# Instance data

   ODULongiteration;                    // number of mouse double–clicks

   ODBoolean    fNeedToExternalize;   // determine part changes
   ...
   //# identify SOM and OpenDoc methods that this class
   //# will re–implement

   Draw,                    // draws part content in the provided facet
   Externalize,             // called when doc in which part resides is saved
   HandleEvent,             // process events, e.g. mouse clicks
   InitPart,                // called during instantiation of a part
   InitPartFromStorage      // called during instantiation of a part from storage
            : override;
   ...
}
```

Hello Component: HELLO.CPP (1)

```
SOM_SCOPE void SOMLINK HelloInitPart   (HelloPart *somSelf,
                                        Environment *ev,
                                        ODStorageUnit *storageUnit
                                    ) {

   HelloPartData *somThis = HelloPartGetData (somSelf);

   HelloPartMethodDebug (”HelloPart”, ”HelloInitPart”);

   HelloPart_parent_ContainerPart_InitPart (somSelf, ev, storageUnit); // init parent

   // Initialize instance data, e.g. somThis–>iteration = 0
   somSelf–>HelloCommonInitPart(ev);

   // ‘‘set dirty”
   somThis–>fNeedToExternalize = kODTrue;

   // prepare part for writing of data by adding a contents property value
   // to the storage unit
   storageUnit–>AddProperty(ev, kODPropContents)–>AddValue(ev, kHelloPart);

};
```

Hello Component: HELLO.CPP (2)

```
SOM_SCOPE void SOMLINK HelloExternalize   (HelloPart *somSelf,
                                           Environment *ev )
{
  // Stores HelloPart part persistent data in part's storage unit
  ...
  // inherited info, e.g. mod date
  HelloPart_parent_ContainerPart_Externalize(somSelf, ev);

  if (somThis–>fNeedToExternalize) {                    // check whether "dirty"

    ODStorageUnit *su = somSelf–>GetStorageUnit(ev);    // get ref to storage

    // focus on HelloPart storage unit
    su–>Focus( ...)

    // write data to storage unit
    su–>SetValue (ev, sizeof(SomThis–>iteration), &somThis–>iteration);

    // reset flag
    somThis–>fNeedToExternalize = kODFalse;
  }
};
```

Hello Component: HELLO.CPP (3)

```
SOM_SCOPE void SOMLINK HelloInitPartFromStorage     (
                                                    HelloPart *somSelf,
                                                    Environment *ev,
                                                    ODStorageUnit *storageUnit // storage unit
                                                    )
{
   // retrieves the part's data from the storage
      ...
   // init parents
   HelloPart_parent_ContainerPart_InitPartFromStorage (somSelf, ev, storageUnit);

   // Init instance data of part, e.g. somThis–>iteration = 0
   somSelf–>HelloCommonInitPart(ev);

   somThis–>fNeedToExternalize = kODFalse; // no need to externalize

   // focus storage unit on the part to be retrieved
   storageUnit–>Focus (ev, ..., HelloPart, ...);

   // read the data from storage unit into the instance variable
   storageUnit–>GetValue(ev, storageUnit–>GetSize(ev), &somThis–>iteration);
};
```